Explanation for Problems

Problem 1

Approach -

We are supposed to find out the new array after increasing the elements that are divisible by 2^x by 2^(x-1).

We take in the main array (v), we take in the queries (vq), and then for each query, we check wether the number is divisible by 2^x. For this, we use this approach for fast calculation - (v[j] & ((1 << vq[i]) - 1)) == 0. This expression checks wether the remainder on dividing v[j] with 2^vq[I] gives remainder 0. If it does, then we decrement j to check at the same index again until the number obtained is not divisible. We do this for every element in the main array and then we print the result.

Time Complexity -

O(nq) as their are two ‘for’ loops.

Space Complexity -

O(1) as we are not making any extra data structures apart from those in the input.

Problem 2

Approach -

We are supposed to find the maximum bitwise AND value from a subarray of any length.

The thing to note here is that taking the AND of two numbers will produce a result that is almost equal to or lesser than the smaller of the values. Hence taking a subarray whose length is greater than one will not give us the desired answer.

Therefore, we just need to find the maximum element in the array.

Time Complexity -

O(n) to find the largest element.

Space Complexity -

O(1) as we are not creating any extra data structures.

P.S - In the first test case {1,2,3}, if we take the subarray to be of length 1, we will get our answer as 3. But in the problem the answer is given as 0. Could you please look into it?

Problem 3

Approach -

Here we need to find two prime numbers, one strictly greater than d (called p1), and the other at least greater than or equal to p1+d.

We create an array of prime numbers up to 10^5 as using sieve of Eratosthenes (as it is given in the question that the maximum d can go is 10^5). This will help us in quickly finding the prime numbers.

We then simply iterate over the array until we find a prime number greater than d, then we iterate to find the other number greater than or equal to p1 + d.

We multiply these two to get the answer.

Time Complexity -

O(nlog(logn)) for creating the Sieve of Eratosthenes. But this will be made only once and will be used for all the test cases.

O(n) for searching through the array.

Space Complexity -

O(n) for creating the Sieve of Eratosthenes.

P.S - In the third test case, d is 3. So p1 should be 5. Now, according to the question, p2 should at least be 8 (5+3). So the smallest prime number greater than 8 is 11. Hence the answer should be 5*11 = 55 and not 35 as given in the pdf. Also, 35 as a product of primes is 5 * 7, which means they took p2 as 7. But p2 must be at least greater than 8. Could you please look into it?

Problem 4

Approach

Here we are supposed to find the number of positions that exist such that both the positions can be reached simultaneously on an infinite grid.

We first generate all possible moves, then we generate all possible previous positions (x,y) that could have reached (xk,yk) by subtracting each movement vector. Then, for each position (x, y) in that set, we apply all movement vectors and check if any reach cq, yq. We then count the result.

Time Complexity -

Since we have a fixed number of moves (8 for each step), hence the time complexity becomes O(1).

Space Complexity -

Again, since the vectors that are being used have fixed sizes, the space complexity becomes O(1).